The five pillars of computational reproducibility: Bioinformatics and beyond

Mark Ziemann, Pierre Poulain, Anusuiya Bora

2023-06-16

Mark Ziemann1\*, Pierre Poulain2, Anusuiya Bora1

1. Deakin University, School of Life and Environmental Sciences, Geelong, Australia.
2. Université Paris Cité, CNRS, Institut Jacques Monod, Paris, France.
* Correspondance: m.ziemann@deakin.edu.au

|  |  |
| --- | --- |
| Author | ORCID |
| Mark Ziemann | 0000-0002-7688-6974 |
| Pierre Poulain | 0000-0003-4177-3619 |
| Anusuiya Bora | 0009-0006-2908-1352 |

Source: <https://github.com/markziemann/5pillars/blob/main/manuscript/5pillars.Rmd>

## Abstract

Computational reproducibility is a simple premise in theory, but is difficult to achieve in practice. Building upon past efforts and proposals to maximize reproducibility and rigor in bioinformatics, we present a framework called the five pillars of reproducible computational research. These include (1) code version control and sharing, (2) compute environment control, (3) literate programming, (4) documentation and (5) FAIR data sharing. These best practices will ensure that computational research work can be reproduced quickly and easily, long into the future. This guide is designed for bioinformatic data analysts, but should be relevant for other domains.

**Keywords:** Reproducibility; computational research; bioinformatics; research best practices.

## Introduction

In research, computational reproducibility is the ability to use the materials from a past study (such as data, code and documentation) to regenerate the outputs including figures and tables to confirm the study’s findings [1]. Reproducibility is only the first step towards overall reliability, for example a study may be reproducible, but suffer from analytical problems that invalidate the results. If a study is reproducible, then at least these analytical issues can be more easily identified and rectified. Therefore, reproducibility is a necessary stepping-stone in the framework of cumulative science. Reproducibility should enhance the overall reliability of computational research including replicability and robustness. Replicability being defined as the ability to repeat the entire study and come up with similar overall results. Robustness refers to the ability for the findings to be generalisable to other situations [2].

Unreliable research is a waste of resources and time and can mislead readers to pursue doomed research directions. Much has been written on how irreproducibility has reached crisis levels in psychology and life sciences [3,4]. Indeed, in bioinformatics, the situation is grim with a 2009 systematic evaluation showing only 2 of 18 articles could be reproduced (11%) [5], bringing into question the reliability of those studies. Workshops conducted by NIH intramural researchers in 2018/2019 sought to reproduce five bioinformatics studies and could not reproduce *any*, citing missing data, software and documentation [6]. A recent systematic analysis of Jupyter notebooks in biomedical articles had similar observations, with only 245/4169 notebooks (5.9%) giving similar results compared to the original, with irreproducibility blamed on missing data, broken dependencies and buggy code [7]. A similar survey of R scripts in the Harvard Dataverse repository found slightly better results, with 26% of scripts completing without errors [8]. Irreproducible bioinformatics has the potential to place patient safety at risk, as demonstrated by an earlier case study [9]. Researchers are theoretically able to achieve 100% computational reproducibility, yet as we see, in practice this is rarely achieved.

A number of guides recommending enhanced computational reproducibility practices have been developed [10–20]. Inspired by these principles, this article presents the *five pillars of reproducible computational research* (**Figure 1**). There are clear parallels with the established three pillars of the open science framework, (open data, code and papers) [21]. Here, the emphasis is on practical reproducibility with an increased focus on programming practices, transparent reporting and the role of computational environments. The intended audience is bioinformatics data analysts (not tool developers), however the principles described here could equally apply to other domains of study.



Figure 1. The five pillars of computational reproducibility.

## End-to-end automated process

For a bioinformatics project to be reproducible, it needs to be enshrined by code and include all instructions, from inspecting the raw data to generating the outputs that form the conclusions of the study. This automated process removes the need for any manual steps, which are time-consuming and prone to errors. Without an end-to-end automated process, most of the other best practices described here are not possible. Scripted workflows, although not always free of errors, enable better auditing and easier reproduction which would be difficult for graphical tools like spreadsheets or webtools. Indeed, spreadsheets are prone to data entry, manipulation and formula errors [22], and are somewhat overused and misused in the life sciences, leading to problems like inadvertent conversion of gene names to dates in databases and supplementary Excel files [23–25]. While webtools are valuable for data exploration, there are worries that they undermine reproducibility for the sake of convenience [26]. Transferring data between compute platforms is also discouraged. For example, having workflows that involve combinations of web-based and scripted tools require data transfer steps which are inefficient and error-prone.

To facilitate an end-to-end automated process, the code and data need to be “linked,” which means the code is aware of the location of the data and how to get it automatically [11]. The code should be able to fetch the data from a publicly accessible location and conduct the downstream processing and analysis without having to acquire the data some other way.

A caveat for end-to-end automation is that sometimes manual data cleaning is unavoidable. In those cases, retaining raw and cleaned data along with a cleaning protocol is recommended. Then the computational workflow can begin with the cleaned dataset.

Another issue is that perfect reproducibility isn’t possible in cases where some element of randomness is requrired by the underlying algorithms. Such algorithms are extensively used in molecular simulation, machine learning, permutation-based statistical tests and certain data projection methods, namely t-distributed stochastic neighbour embedding (t-SNE) [27] and uniform manifold approximation and projection UMAP [28], which are popular for visualising high-dimensional omics data [29]. To make such workflows deterministic, the pseudo-random number generator can be initialized with a fixed value (sometimes called “setting the seed”)[12]. However this needs to be done with care to ensure the results do not misrepresent the bulk of iterations (see [30]).

A guiding principle of the five pillars approach is that the publishing researchers should strive to make the reproduction process easier and faster for those who wish to. A “master script” that coordinates the execution of individual parts of an analysis is an excellent way to reduce the number of commands required for reproduction. This can be coded in standard open source languages like R/R Markdown, Python/Jupyter notebook or the Unix Shell. Using free and open source software ensures that the research is accessible to the greatest audience [31], as opposed to proprietary software like SPSS, STATA and MatLab which are cost inhibitory. Free and open source software also enjoy relatively larger libraries of add-on packages contributed by the scientific community. Projects entailing big data analysis might benefit from a build/workflow automation solution like make, snakemake [32,33], targets [34] or nextflow [35]. The advantage of such tools is that if the analysis were to terminate due to an issue mid-way, for example, a hardware problem at step 8 of a 15-step workflow, the analysis of steps 1-7 wouldn’t need to be repeated. After fixing the hardware issue, re-running the analysis would pick up at step 8 again, which saves labor and compute time.

## Literate programming

Literate programming combines “chunks” of analytical code with human readable text [36]. After compilation, the resulting output document contains the code together with computational results such as figures and tables along with contextualizing explanations and narratives.

The Sweave project, conceived in 2001, was designed to give LaTeX documents embedded R code chunks and was envisaged as a way to generate R documentation materials and generate statistical analysis reports [37]. Around 2015, R Markdown emerged as a more popular alternative, as formatting content is simpler with Markdown as compared to LaTeX, saving time. An example R Markdown script and output report is given in **Figure 2**.

![Figure 2. An example R Markdown script (left) and output document (right) [38].]()

Figure 2. An example R Markdown script (left) and output document (right) [38].

Around the same time, Project Jupyter was developed to provide a “notebook” type interface, incorporating R, Python, Julia and other computer language chunks into documents [39–41]. A noteworthy attempt to use literate programming in Jupyter notebooks to aim for reproducibility has been the analysis of RNA-seq samples from patients infected by the Zika virus [42]. In this work, authors provided along the traditional research paper a [Jupyter notebook](https://nbviewer.org/github/maayanlab/Zika-RNAseq-Pipeline/blob/master/Zika.ipynb) that performed and documented the entire analysis procedure. The MyST-NB (short for Markedly Structured Text notebook) is built on top of Jupyter notebooks or Markdown documents and has some notable advantages around flexibility of content that make it an attractive choice for executable article authoring [20].

In 2022, the company behind RStudio (Posit) released Quarto, the conceptual successor of R Markdown, but with enhanced support for other computer languages like Python, Julia, and Observable JavaScript [43].

Whether R Markdown, Jupyter or other system, literate programming offers some substantial benefits over alternative approaches:

* The provenance of any result can be demonstrated. The resulting document produced contains the code executed, together with the results (e.g.: a chart). This is in contrast to a data analysis report assembled in a word processor, which due to the many copy-paste operations required might contain errors, or version mismatches.
* It saves time. For the case where a report needs to be run routinely, this would involve significant copy-paste to be assembled “manually” with a word processor. Using a literate script would mean only minimal changes are required between iterations [44].
* It accommodates extensive documentation. This allows the analyst to include rich descriptions of scientific works. For example, a report may contain background, methods, results, discussion and references. Not only text, but various objects can be included like links to other resources, tables, images, videos, etc. This means it is possible to author an entire journal article using literate programming.
* Outputs are arranged. When using a regular script, an analyst might make dozens of charts which are written to the disk with names like “chart1.png” or “model8.svg.” When the number of outputs is large, it becomes unwieldy and hard to place which plot corresponds to which part of the script. By embedding the outputs such as charts and tables in a document in the sequence that they were generated, it helps the reader understand the logical steps taken in an analysis.
* Amenable to version control. Version control is a useful best practice in software development (discussed below).
* Output reports are free from code errors. The output document is only rendered when the entire script has been compiled without errors. In contrast, a regular script might generate some outputs and then encounter an error, so we are not sure whether the whole script is free of errors and completed successfully. Therefore, it is good practice to routinely execute these scripts during the development process, and not rely too heavily on the interactive execution of individual chunks or lines.
* Flexible output formats. These include PDF, DOC/DOCX and HTML, the latter having some notable benefits including better usability for mobile devices like phones and tablets, and the ability to support richer content such as interactive charts and dynamic tables (searchable, filterable, sortable). Outputs include HTML documents, but also slideshows and even ebooks.

These features make literate programming a quantum leap for science communication in a range of situations. Whether this is sharing a data analysis report, giving a presentation at a meeting, writing a research article or self-publishing an ebook, literate programming provides the ability to construct transparent data narratives with clear provenance in a conveniently shareable form.

Literate programming also works neatly with the concept of the “executable paper,” the idea that the data analysis underlying an entire study can be reproduced with one or a few commands [45,46,47]. A typical genomics/bioinformatics study could involve one large literate script, or be broken down into smaller scripts, where each one contains the code for generating a part of the article. A multi-script approach would benefit from a master script that executes each component. This further makes the job of reproducibility easier.

## Code version control and persistent sharing

A version control system (sometimes called “source control”) is a type of program that “tracks changes” made to sets of files, typically other computer program source code, scripts and documentation. A set of files under version control is called a “repository” and represents a project or sub-project. Version control is used extensively by software developers and is considered one of the key best practices in software engineering. Distributed version control systems (DVCS) involve a central web-accessible server hosting a repository, and each team member possesses a mirror copy on their local system (**Figure 3**). Having a central, publicly available node assists in disseminating changes within teams and releasing code to consumers. There are many such DVCSs (Subversion, Git, mercurial, etc), but git has emerged as the most popular solution due to its many powerful features, speed/efficiency and large community and ecosystem.

![Figure 3. Distributed version control. Adapted from [48].]()

Figure 3. Distributed version control. Adapted from [48].

Computational researchers can benefit significantly by adopting DVCS into their routine practices. Below we outline some of the benefits.

* Keeps a complete history of all code changes over time. We always know which version of the script is the most up to date. We can inspect and execute the code and reproduce the analysis at any particular point in the past. Commit messages are retained, which gives a record of the motivations, purpose and person behind each modification.
* Helps collaboration and project management. Using a centralized code hosting platform such as “GitHub” or “BitBucket” can help manage contributions from team members who may be working asynchronously in different time zones and on different servers. It also makes the user’s work searchable, which is convenient when code snippets from a project need to be reused for a new project.
* Helps resolve issues. These centralized platforms also enable discussions of code issues such as bugs and feature requests. In bioinformatics teams, issues can be used to track the progress of a project through its milestones and allow team members to raise potential issues with the code. This also allows the scientific community to raise an issue with the software maintainers. For example, this feature is used extensively during the peer review process for the *Journal of Open-Source Software* [49].
* Supports documentation best practices. Books, tutorials, protocols and other educational resources are well served using DVCS (see below).
* Makes works easily shareable. Work is easier to reproduce because the source folder contains a complete workflow with code linked to data and metadata, so it doesn’t need any modifications before reproduction. It is relatively easy to add a software license with a DVCS like GitHub, so consumers can understand the conditions of using it.
* Protects against code loss. Computers sometimes fail and we sometimes delete or overwrite important files. If code is lost and unrecoverable it can be a significant cost to replace. Although DVCSs are not a back-up solution, they do add a layer of protection to the code. Preserving a copy of the repository on each team members’ computer in addition to the central repository means the code can be easily recovered if any one of those computers fails.

Although DVCSs assist with code sharing, they are not considered a long-term archiving solution, as recalled by the complete shutdown of the Google Code platform in 2015 and the end of Mercurial support by the Bitbucket platform in 2020. The code needs to be deposited to a long-term repository. [Zenodo](https://zenodo.org/) and [Figshare](https://figshare.com/) are often used to store the content of a repository but do not retain the history of the development of the source code. [Software Heritage](https://www.softwareheritage.org/) [50,51] is the universal software archive whose mission is to collect, preserve and share source code. Software Heritage provides an intrinsic persistent identifier called SWHID that allows to cite the archived source code in the respective journal article/preprint [52]. Archiving source code in Software Heritage can be performed [manually](https://archive.softwareheritage.org/save/) or automatically using a [webhook](https://docs.softwareheritage.org/user/save_code_now/webhooks/index.html) within continuous integration.

git is typically used at the command line, however it is also integrated into Rstudio, JupyterLab and VS Code, and there are also several git clients with graphical interfaces which better allow inspection of code changes.

Finally, although DVCSs come from the world of software engineering, their use has become a standard in computational biology, as witnessed by the numerous guides and best practices on the subject [53–55].

## Compute environment control

Most popular software undergoes regular updates to patch bugs and add new features. Bioinformatics software is no different, and it is well known that such changes have the potential to affect results [56]. This is why it is best practice to report the exact version of all programs used in an analysis (and packages therein) and even make archival copies for future reference [12].

In R, such reporting is possible using the sessionInfo() command, while for Python this is possible using the session\_info or watermark packages. Using literate programming and sharing output documents ensures that a record of this important information is made available.

Although regular software updates are overall a good thing, it poses a problem for future reproducibility. A researcher trying to reproduce a ten-year-old study in R v3.0 could have a headache, as they would need to roll back their R version, and possibly their operating system as well, as R and other languages require certain system dependencies for low-level routines [57].

To avoid this, a virtual machine (VM) could be used to run a system-in-a-system. This means that the “host” machine can run another “guest” operating system with the right R version, without needing to change the host R version. While this provides good reproducibility, setting up an environment with a ten-year-old operating system, R and packages would take a few hours to accomplish. Researchers could take a snapshot of their VM system and share it to help reproducibility and auditability [58,59], however the size of these images is relatively large due to the fact it contains the OS, software stack and project data. Moreover, the performance of computation in the guest system is typically slower than when run directly on the host.

Containers are an attempt to resolve some of the downsides of VMs. Container images can be thought of as similar to VMs, but are more lightweight as they share parts of the host operating system [60]. In the example shown in **Figure 4**, the five containerised applications share the same operating system, while the three VM applications each involve their own operating system which incurs a significant performance overhead [61]. Therefore, running workflows in containers incurs only a small reduction in performance as compared to running directly on the host system [62]. Container images are highly portable because containers include everything needed to run the application, including the system tools and libraries, ensuring that the environment is consistent across different systems. For example, the most popular containerisation system, Docker [63], makes it possible to run Windows and Linux/Unix containers on any computer with Docker installed, with the promise of reproducibility. While there are several alternatives such as Podman, Rkt or Apptainer/Singularity, Docker remains the most widely used containerisation system for web development and research. Docker now has a large community of users, extensive documentation and a vast collection of pre-built container images in the DockerHub registry. Docker can fetch images from DockerHub and run them on the host system with just a couple of commands, and typically within a few minutes. This accelerates the installation procedure dramatically which is a known bottleneck for bioinformatic reproducibility [64,65].

![Figure 4. Architecture of containers (left) compared to VMs (right). Adapted from [61].]() Another potential solution to this problem is to use a package/environment management system such as [Conda](https://conda.org/) or [Guix](https://guix.gnu.org/). These allow users to create, manage, and deploy software packages, dependencies, and environments across different computing platforms [66]. Conda was initially developed to solve the problem of package dependency management in Python, but it now supports many other programming languages, including R, C/C++, Java, and others. Conda allows researchers to create isolated environments with specific versions of packages, so users can have different versions of R or Python in different environments on the same host system. Conda environments have limitations around portability and consistency, as such environments are tied to specific operating systems and hardware architecture.

Guix is described as a “functional package manager” and has the unique benefit of giving bit-for-bit build reproducibility and verifiability [57,67]. This is a notable advantage, as Docker images are not verifiable nor guaranteed to build reproducibly in the future due to link decay. Moreover, Guix can build and output environments as Docker-compatible images which solves the provenance problem of such environments [68]. These features are being used in the bioinformatics sphere to create highly reproducible pipelines that work equally well on personal as well as shared high performance computers [67,69]. Guix is among the best solutions for the “ten-year challenge” of code reproducibility [70]. Although Guix has some exciting functionality, there remains a relative lack of step-by-step guides and tutorials, illustrated by the complete absence of published Guix protocols in protocols.io, as compared to 13 and 12 related to “Conda” and “Docker” respectively (as at 30th May 2023). Lastly, it’s worth mentioning that the Guix project uses Software Heritage as a fallback system to provide a package source code if the download from its original location failed.

We are also watching with interest the early development of WebAssembly (Wasm) based approaches for compute environment control. The ability to reproduce operating systems, programming languages and workflows in the browser opens up intriguing possibilities for more widespread reproduction and auditing without the need to install any software at all. Two notable examples of this include JupyterLite, a version of JupyterLab in early development that runs entirely in the browser [71], and WebR-enabled interactive code blocks in Quarto HTML documents [72].

It is not sufficient to simply use these tools, they need to be shared as part of the publication process. Conda environments are described by a file called environment.yml which acts like a recipe for how to make the environment. Similarly, Guix environments rely on two files: channels.scm and manifest.scm. Docker images are built using a Dockerfile instruction set. These are small files which are easily shared in the project code repository. To enable rapid reproduction, making Docker images available is recommended. As these are often very large they cannot be shared in the code repository; rather they are commonly deposited to a Docker image registry such as DockerHub. One must keep in mind that availability of images is dependent on the commercial viability of Docker Inc, so it isn’t guaranteed to survive the 10-year challenge. To ensure long-term availability, it would be advised to deposit images to the BioContainers registry [73], as it is community driven and supported by consortia such as *Elixir* and *Global Alliance for Genomics and Health* that will underpin its longevity. Still, these are not considered permanent archival services, so it is advised to deposit the image used in an article to a suitable persistent long-term archive (discussed below).

## FAIR and persistent data sharing

Without data sharing, computational research is not reproducible nor auditable. Lack of data sharing is one of the key reasons why research is irreproducible [74]. Data sharing is also one of the key features of “open science,” which is characterized as science that is collaborative, transparent, accessible and inclusive [75,76]. In addition to facilitating reproduction and auditing, sharing enables reuse in other contexts. Data reuse increases efficiency, as it prevents redundant research expenditure and facilitates new research ideas that were previously impossible [77]. In terms of research rigor, inspecting raw data can uncover inadvertent errors and research integrity problems [78]. In light of this, a research article without supporting data and code is much like a press release or advertisement where the claims made cannot be verified as true or not [20].

Although it is common to see journal articles with “Data available upon reasonable request,” this is less than ideal. A systematic study of such data availability statements found that the data was successfully provided in just 6.8% of requests [80]. In the rare cases that data is shared in accordance with the data availability statement, it poses a burden in terms of labor to lodge and respond to such requests, and if the data is large, may incur an additional cost for hosting/transferring these large files.

The field of genomics has a long history of data sharing [81], which is supported by policy leadership by funding bodies [82,83] and data sharing policies of journals (eg [84,85,86]), although there are some challenges. Some researchers would rather keep data for their private use. Some may keep data private due to concerns over participant privacy. This is a well-founded concern for identifiable data, however once identifying features are removed, it is nearly impossible to re-identify genomic data [87] unless the attackers have pre-existing genomic data describing an individual or ancestry [88,89]. Therefore, sharing de-identified genomic data and metadata is likely to carry little risk to participant privacy. Still, much human genomic data and metadata resides in restricted databases due to the scope of data sharing that was agreed to in the consent process. For example, the consent form may only allow sharing of data to researchers investigating certain diseases, which means a vetting process must be undertaken before sharing data with external investigators. In the interests of reuse and reproducibility, future studies could consider a consent form that allows more broad sharing of de-identified data.

In order to maximize the value of shared data for reuse and reproducibility, it needs to be Findable, Accessible, Interoperable and Reusable (FAIR) for people and for computers [77]. These principles are summarized in **Figure 5**.

![Figure 5. Overview of the FAIR data principles [90].]()

Figure 5. Overview of the FAIR data principles [90].

High throughput biology has already built a number of data repositories for specific data types conforming to FAIR principles (e.g.: GEO, SRA, ENA, PRIDE [91–93]). However other domains lack well known specialized organized data repositories, such as evolutionary biology, ecology, cell biology, molecular simulation, social sciences and computing. The [re3data.org](https://www.re3data.org/) registry of data repositories may be useful to find a repository that accepts data from these fields [94]. If no specialized repository exists, then a general-purpose repository such as [Dryad](https://datadryad.org/), [FigShare](https://figshare.com/), or [Zenodo](https://zenodo.org/) should be considered. Researchers should be aware, however, that these repositories are often not moderated, and that it is their responsibility to make their data FAIR, in particular by providing sufficient metadata. To this aim, general recommendations have been proposed to enhance data FAIRness in ecology, but which can also be applied to other disciplines [95].

Key recommendations include:

1. Depositing data to a specialized repository if possible, otherwise, a general-purpose repository.
2. Avoid commodity cloud storage as these are impermanent and susceptible to link decay [96].
3. Avoid large supplementary data files accompanying journal articles as these are less findable and accessible.
4. Preferably archive and share raw data and use existing standards for the discipline.
5. Use file formats that are compatible with many different types of software; e.g.: comma-separated values (CSV) files are preferred over Excel files.
6. Provide detailed metadata; e.g.: sample descriptions that match the article; describe the columns in tabular data (i.e.: data dictionary).

Once the quality of the data set has been established researchers may consider depositing it to a data repository early, before publicization, as it has some advantages. Most data repositories have an optional 12-month embargo period so researchers can share publicly at a later date once they’re happy with the overall project. The data repository acts as an additional backup to the host institutions’ own in case of data loss or calamity. By assuming the data and metadata can be sourced from a public location, the workflow developers ensure that the code they share won’t suffer from “file not found errors” during reproduction. This ensures code and data are linked, which has been a priority for reproducibility [11], and a source of problems for shared code [7].

In addition to the experimental data, reference data that is critical to a project should be archived. For example, in the field of genomics, gene function annotation sets are relied upon for a common procedure called “pathway analysis” [97]. These annotation sets are regularly updated, but versions are rarely reported [26] and finding historical data versions is sometimes difficult. In order to guarantee future reproducibility, snapshots of those reference data should be archived and shared if the license permits it.

## Documentation

Documentation is the glue that binds a data science project together. The published article is the central artifact that outlines the research project and links to the supporting materials. From a reproducibility standpoint, the methods section is the most critical part. It should be detailed enough so that other researchers can understand and replicate the experiments/analysis and yield similar results. Unfortunately, key details in bioinformatics data processing procedures are often omitted which limits their reproducibility [31,98,99]. Commonly missing information includes versions of software and packages as well as any parameter setting and configuration files. The “Materials Design Analysis Reporting” (MDAR) checklist for authors has been developed to assist in comprehensive methodological reporting in the life sciences [100], and “Minimum Information About a Bioinformatics Investigation” guidelines describe good reporting practices for computational biology research [10].

Researchers should consider depositing their thorough laboratory/*in silico* protocols as separate outputs to relevant repositories such as [protocols.io](https://www.protocols.io/), [RIO Journal](https://riojournal.com/) or [Zenodo](https://zenodo.org/), minting a Digital Object Identifier (DOI) that can be cited in the article. This is particularly helpful when there are strict word limits on articles.

The article should have clear links to supporting materials including datasets, software code and other resources like computational environments.

The code repository should have a detailed README file which plays a critical role in reproducibility. It should outline the purpose of the code/software/overall project and how it relates to the article. For example, some articles may rely on more than one repository for various parts of the research project, so these need to be explained.

The README should outline exactly what is required in order to reproduce the analysis, including the requirements and the instructions to reproduce. Typically it is written in the Markdown format, which should be familiar to those using R Markdown or Jupyter notebooks. Hardware requirements such as RAM, CPU architecture, GPU needs, etc need to be outlined. Software requirements, such as operating system, dependencies, container engine, workflow manager, etc, also need to be described. The exact instructions for reproduction should be clearly outlined. This should include what the output objects of the analysis are, and instructions on how to access the results and what should be expected. Literate programming allows for thorough documentation of workflow methods and results which makes it more accessible for reproducers to comprehend the workflow details, which is not possible with standard code comments [101]. These instructions should be tested whenever changes are made to the codebase.

As outlined above, one of the goals is to reduce the complexity of reproduction, and using a master script is a simple way to do this. It also makes the README documentation much simpler. For example, the pathway analysis workflow we previously developed could be reproduced and inspected with just six commands, starting from a new Ubuntu Linux installation [102]. By minimizing the complexity of reproduction and writing the instructions clearly in the README, reproducibility is made possible to a larger segment of the computational research community, especially those who are not experts in the languages used.

In addition, the README should outline the contents of the repository, how users can contribute to the codebase and how to report issues, such as code errors and discrepancies. The README should also list the contributors to the codebase and acknowledge any financial support. It may also include a link to the research article, citation instructions and a mention of the software license.

The need for documentation extends to the other supporting information. The deposited data set needs thorough descriptive metadata to provide context. The container image and code snapshot should be deposited to a persistent repository with a thorough description of their purpose. Both the data and container image descriptions need to contain links to the DOI of the article, so that machines or individuals browsing these resources can understand the context of these objects relative to the overall research project.

## Limitations and challenges

Due to the constantly developing software landscape, this review is not intended to give strong recommendations on the exact combination of tools or resources to use in computational research, but to set out key principles and highlight some useful solutions that users can incorporate into their toolkit to improve the overall reproducibility of their work.

When these principles are not integrated correctly and checked, problems readily emerge [6,7,103]. This is why regular code testing is another recommended best practice, especially after making updates to the code or data [101,104]. It is worthwhile to conduct such testing on a separate computer to understand how someone else in the scientific community would fare reproducing such a workflow.

Moreover, some have suggested adapting software development principles of unit tests and continuous integration/continuous delivery. This approach, called “continuous analysis” involves automatic execution and testing whenever a change is made to a project’s code base [105,106]. If the workflow executes successfully and tests are passed, the repository can be decorated with a “badge” to inform users of the lack of errors in the code. If those tests are designed cleverly such that certain conclusions drawn in the corresponding research article can be assessed computationally, then such tests could be used to demonstrate computational reproducibility without the need for individuals to execute the workflow themselves. Continuous analysis might therefore be a future pediment (crowning feature) to the five-pillared “temple” of reproducible research.

The key technological advances enabling the five pillars approach have been around for about a decade, yet progress on integrating these into computational research publications has been minimal. Indeed the number of systematic replication studies in bioinformatics remains small [5–7], so the severity of the problem isn’t well understood. Part of the problem is the incentive structures in academia, with funding bodies and journals largely focused on novel research directions rather than confirming previous findings [31,107–110].

A lack of technical computer skills in many research groups is another culprit. Specific workshop-based training in these areas could help [111]. The [Software Carpentry](https://software-carpentry.org/) initiative is focused on this mission [112]. However, the most scalable solution would be to provide comprehensive beginner-friendly step-by-step guides for frequently used routines. For example, our group has just released such a protocol that walks novice users through the process of implementing functional enrichment analysis of gene expression data with the five pillars principles [102]. More such protocols are needed to make extreme reproducibility mainstream. A compendium of publicly available learning materials around the five pillar concepts are provided in the supplementary information.

## Conclusion

Reproducibility is an essential feature of reliable computational research yet current practices are falling short of this goal. Our recommendations for computational researchers are summarized as:

1. Encapsulate the entire workflow in an end-to-end automated process, executable with a master script.
2. Share the code repository and deposit a snapshot to an archive.
3. Share a controlled environment; options include a Docker or virtual machine image or configuration for Conda or Guix.
4. Use literate scripting tools like R Markdown or Jupyter notebooks to generate all outputs, such as charts and tables.
5. Document the workflow sufficiently for reproduction, especially the repository README.
6. Share research data and metadata according to FAIR principles.

## Key points

* Irreproducibility of bioinformatics studies remains a significant and still relevant problem.
* We present the *five pillars* framework, a set of best practices that enable extremely reproducible workflows.
* Widespread adoption of these principles will enhance research reliability and will speed translation of basic research to tangible benefits.

## Acknowledgements

We are grateful to individuals who provided feedback on an earlier draft of the concept including Dr Aziz Khan (Stanford University), Dr Sriganesh Srihari (QIMR Berghofer Medical Research Institute), Dr Barbara Zdrazil (EMBL-EBI) and Dr Irene López Santiago (EMBL-EBI). Special thanks to Dr Altuna Akalin (Max Delbrück Center, Germany) and Dr Simon Tournier (Paris Cité University, France) for advice on Guix. We appreciate advice on Python notebooks from Samuel S. Minot (Fred Hutchinson Cancer Center). We are grateful to Dr Martin O’Hely (Deakin University) for critical comments on the manuscript. This research was supported by use of the Nectar Research Cloud, a collaborative Australian research platform supported by the NCRIS-funded Australian Research Data Commons (ARDC).

## Funding

The authors received no specific funding for this work.

## Availability of materials

Code repository: <https://github.com/markziemann/5pillars>

## Supplementary information

[Supplementary materials 1: A curated compendium of publicly available educational materials related to five pillar concepts.](https://ziemann-lab.net/public/5pillars/practical_guides.html)

## Author biographies

**Mark Ziemann** is a Senior Lecturer in Biotechnology and Bioinformatics at Deakin University. His research spans bioinformatics as applied to omics data analysis in biomedicine, as well as aspects of research rigor.

**Pierre Poulain** is an Associate Professor at Université Paris Cité, Institut Jacques Monod, France. Research directions include bioinformatics, large scale data analysis and education/pedagogy.

**Anusuiya Bora** is a PhD student at Deakin University School of Life and Environmental Sciences. Her research project involves improving the reproducibility of commonly used bioinformatics tools.

## Bibliography

1. Goodman SN, Fanelli D, Ioannidis JPA. What does research reproducibility mean? Sci Transl Med. 2016;8: 341ps12–341ps12.

2. Nosek BA, Hardwicke TE, Moshontz H, Allard A, Corker KS, Dreber A, et al. Replicability, robustness, and reproducibility in psychological science. Annu Rev Psychol. 2022;73: 719–748.

3. Baker M. First results from psychology’s largest reproducibility test. Nature. 2015. doi:[10.1038/nature.2015.17433](https://doi.org/10.1038/nature.2015.17433)

4. Begley CG, Ellis LM. Drug development: Raise standards for preclinical cancer research: Drug development. Nature. 2012;483: 531–533.

5. Ioannidis JPA, Allison DB, Ball CA, Coulibaly I, Cui X, Culhane AC, et al. Repeatability of published microarray gene expression analyses. Nat Genet. 2009;41: 149–155.

6. Zaringhalam M, Federer L. Data and code for reproducible research: Lessons learned from the NLM reproducibility workshop. Zenodo; 2020. doi:<https://doi.org/10.5281/zenodo.3818329>

7. Samuel S, Mietchen D. Computational reproducibility of jupyter notebooks from biomedical publications. arXiv. 2022; 2209.04308.

8. Trisovic A, Lau MK, Pasquier T, Crosas M. A large-scale study on research code quality and execution. Sci Data. 2022;9: 60.

9. Baggerly KA, Coombes KR. Deriving chemosensitivity from cell lines: Forensic bioinformatics and reproducible research in high-throughput biology. 2010; 1309–1334.

10. Tan TW, Tong JC, Khan AM, Silva M de, Lim KS, Ranganathan S. Advancing standards for bioinformatics activities: Persistence, reproducibility, disambiguation and minimum information about a bioinformatics investigation (MIABi). BMC Genomics. 2010;11 Suppl 4: S27.

11. Peng RD. Reproducible research in computational science. Science. 2011;334: 1226–1227.

12. Sandve GK, Nekrutenko A, Taylor J, Hovig E. Ten simple rules for reproducible computational research. PLoS Comput Biol. 2013;9: e1003285.

13. Piccolo SR, Frampton MB. Tools and techniques for computational reproducibility. Gigascience. 2016;5. doi:<https://doi.org/10.1186/s13742-016-0135-4>

14. Lewis J, Breeze CE, Charlesworth J, Maclaren OJ, Cooper J. Where next for the reproducibility agenda in computational biology? BMC Syst Biol. 2016;10. doi:<https://doi.org/10.1186/s12918-016-0288-x>

15. Grüning B, Chilton J, Köster J, Dale R, Soranzo N, Beek M van den, et al. Practical computational reproducibility in the life sciences. Cell Syst. 2018;6: 631–635.

16. Akalin A. Scientific data analysis pipelines and reproducibility. 2018. Available: <https://towardsdatascience.com/scientific-data-analysis-pipelines-and-reproducibility-75ff9df5b4c5>

17. Brito JJ, Li J, Moore JH, Greene CS, Nogoy NA, Garmire LX, et al. Recommendations to enhance rigor and reproducibility in biomedical research. Gigascience. 2020;9. doi:<https://doi.org/10.1093/gigascience/giaa056>

18. Peikert A, Brandmaier AM. A reproducible data analysis workflow with r markdown, git, make, and docker. Quant Comput Methods Behav Sci. 2021;1: e3763.

19. Rodrigues B. Building reproducible analytical pipelines with R. Leanpub; 2023. Available: <https://raps-with-r.dev>

20. DuPre E, Holdgraf C, Karakuzu A, Tetrel L, Bellec P, Stikov N, et al. Beyond advertising: New infrastructures for publishing integrated research objects. PLoS Comput Biol. 2022;18: e1009651.

21. Gorgolewski KJ, Poldrack RA. A practical guide for improving transparency and reproducibility in neuroimaging research. PLoS Biol. 2016;14: e1002506.

22. Panko RR, Halverson RP. Spreadsheets on trial: A survey of research on spreadsheet risks. Proceedings of HICSS-29: 29th hawaii international conference on system sciences. IEEE; 1996. pp. 326–335 vol.2.

23. Zeeberg BR, Riss J, Kane DW, Bussey KJ, Uchio E, Linehan WM, et al. Mistaken identifiers: Gene name errors can be introduced inadvertently when using excel in bioinformatics. BMC Bioinformatics. 2004;5: 80.

24. Ziemann M, Eren Y, El-Osta A. Gene name errors are widespread in the scientific literature. Genome Biol. 2016;17: 177.

25. Abeysooriya M, Soria M, Kasu MS, Ziemann M. Gene name errors: Lessons not learned. PLoS Comput Biol. 2021;17: e1008984.

26. Wijesooriya K, Jadaan SA, Perera KL, Kaur T, Ziemann M. Urgent need for consistent standards in functional enrichment analysis. PLoS Comput Biol. 2022;18: e1009935.

27. Maaten L van der, Hinton G. Visualizing data using t-SNE. J Mach Learn Res. 2008;9: 2579–2605.

28. McInnes L, Healy J, Melville J. UMAP: Uniform manifold approximation and projection for dimension reduction. 2018.

29. Becht E, McInnes L, Healy J, Dutertre C-A, Kwok IWH, Ng LG, et al. Dimensionality reduction for visualizing single-cell data using UMAP. Nat Biotechnol. 2018;37: 38–44.

30. Benureau FCY, Rougier NP. Re-run, repeat, reproduce, reuse, replicate: Transforming code into scientific contributions. Front Neuroinform. 2017;11: 69.

31. Garijo D, Kinnings S, Xie L, Xie L, Zhang Y, Bourne PE, et al. Quantifying reproducibility in computational biology: The case of the tuberculosis drugome. PLoS One. 2013;8: e80278.

32. Köster J, Rahmann S. Snakemake–a scalable bioinformatics workflow engine. Bioinformatics. 2012;28: 2520–2522.

33. Mölder F, Jablonski KP, Letcher B, Hall MB, Tomkins-Tinch CH, Sochat V, et al. Sustainable data analysis with snakemake. F1000Res. 2021;10: 33.

34. Landau WM. The targets r package: A dynamic make-like function-oriented pipeline toolkit for reproducibility and high-performance computing. Journal of Open Source Software. 2021;6: 2959. doi:[10.21105/joss.02959](https://doi.org/10.21105/joss.02959)

35. Di Tommaso P, Chatzou M, Floden EW, Barja PP, Palumbo E, Notredame C. Nextflow enables reproducible computational workflows. Nat Biotechnol. 2017;35: 316–319.

36. Knuth DE. Literate programming. The Computer Journal. 1984;27: 97–111.

37. Leisch F. The R journal: Sweave, part i: Mixing R and LaTeX. R News. 2002;2: 28–31.

38. Grolemund G, Wickham H. R for data science. Sebastopol, CA: O’Reilly Media; 2017. p. 425.

39. Kluyver T, Ragan-Kelley B, Pérez F, Granger B, Bussonnier M, Frederic J, et al. Jupyter Notebooks a publishing format for reproducible computational workflows. Positioning and Power in Academic Publishing: Players, Agents and Agendas. 2016. pp. 87–90.

40. Granger BE, Perez F. Jupyter: Thinking and Storytelling With Code and Data. Computing in Science & Engineering. 2021;23: 7–14. doi:[10.1109/MCSE.2021.3059263](https://doi.org/10.1109/MCSE.2021.3059263)

41. Perkel JM. Why jupyter is data scientists’ computational notebook of choice. Nature. 2018;563: 145–146.

42. Wang Z, Ma’ayan A. An open RNA-Seq data analysis pipeline tutorial with an example of reprocessing data from a recent Zika virus study. F1000Research. 2016;5: 1574. doi:[10.12688/f1000research.9110.1](https://doi.org/10.12688/f1000research.9110.1)

43. Bauer PC, Landesvatter C. Writing a reproducible paper with RStudio and quarto. OSF Preprints. 2023. doi:<https://doi.org/10.31219/osf.io/ur4xn>

44. Perkel JM. Cut the tyranny of copy-and-paste with these coding tools. Nature. 2022;603: 191–192.

45. Strijkers R, Cushing R, Vasyunin D, Laat C de, Belloum ASZ, Meijer R. Toward executable scientific publications. Procedia Comput Sci. 2011;4: 707–715.

46. eLife Editorial Board. Innovation: Collaborating to devise the future executable article. eLife. 2017. Available: <https://elifesciences.org/inside-elife/c8ca56de/innovation-collaborating-to-devise-the-future-executable-article>

47. Lasser J. Creating an executable paper is a journey through open science. Commun Phys. 2020;3. doi:<https://doi.org/10.1038/s42005-020-00403-4>

48. Chacon S, Straub B. Pro git. 2nd ed. Elk Grove, CA: Apress; 2023. p. 19.

49. JOSS Editorial Board. Editorial guide. <https://joss.readthedocs.io/en/latest/editing.html>; 2018.

50. Di Cosmo R, Zacchiroli S. Software Heritage: Why and How to Preserve Software Source Code. Proceedings of the 14th International Conference on Digital Preservation, iPRES 2017. Japan; 2017.

51. Abramatic J-F, Cosmo RD, Zacchiroli S. Building the universal archive of source code. ACM, editor. Communications of the ACM. 2018;61: 29–31. doi:[10.1145/3183558](https://doi.org/10.1145/3183558)

52. Cosmo RD. Archiving and referencing source code with software heritage. ICMS. Springer; 2020. pp. 362–373. doi:[10.1007/978-3-030-52200-1\_36](https://doi.org/10.1007/978-3-030-52200-1_36)

53. Ram K. Git can facilitate greater reproducibility and increased transparency in science. Source Code for Biology and Medicine. 2013;8: 7. doi:[10.1186/1751-0473-8-7](https://doi.org/10.1186/1751-0473-8-7)

54. Blischak JD, Davenport ER, Wilson G. A Quick Introduction to Version Control with Git and GitHub. PLOS Computational Biology. 2016;12: e1004668. doi:[10.1371/journal.pcbi.1004668](https://doi.org/10.1371/journal.pcbi.1004668)

55. Perez-Riverol Y, Gatto L, Wang R, Sachsenberg T, Uszkoreit J, Leprevost F da V, et al. Ten Simple Rules for Taking Advantage of Git and GitHub. Markel S, editor. PLOS Computational Biology. 2016;12: e1004947. doi:[10.1371/journal.pcbi.1004947](https://doi.org/10.1371/journal.pcbi.1004947)

56. Seyednasrollah F, Laiho A, Elo LL. Comparison of software packages for detecting differential expression in RNA-seq studies. Brief Bioinform. 2015;16: 59–70.

57. Vallet N, Michonneau D, Tournier S. Toward practical transparent verifiable and long-term reproducible research using guix. Sci Data. 2022;9: 597.

58. Hurley DG, Budden DM, Crampin EJ. Virtual reference environments: A simple way to make research reproducible. Brief Bioinform. 2015;16: 901–903.

59. Pasquier T, Lau MK, Han X, Fong E, Lerner BS, Boose ER, et al. Sharing and preserving computational analyses for posterity with encapsulator. Comput Sci Eng. 2018;20: 111–124.

60. Nüst D, Sochat V, Marwick B, Eglen SJ, Head T, Hirst T, et al. Ten simple rules for writing dockerfiles for reproducible data science. PLoS Comput Biol. 2020;16: e1008316.

61. Docker Inc. What is a container? 2021. Available: <https://www.docker.com/resources/what-container>

62. Di Tommaso P, Palumbo E, Chatzou M, Prieto P, Heuer ML, Notredame C. The impact of docker containers on the performance of genomic pipelines. PeerJ. 2015;3: e1273.

63. Merkel D. Docker: Lightweight linux containers for consistent development and deployment. Linux J. 2014;2014: 2.

64. Mangul S, Martin LS, Eskin E, Blekhman R. Improving the usability and archival stability of bioinformatics software. Genome Biol. 2019;20: 47.

65. Mangul S, Mosqueiro T, Abdill RJ, Duong D, Mitchell K, Sarwal V, et al. Challenges and recommendations to improve the installability and archival stability of omics computational tools. PLoS Biol. 2019;17: e3000333.

66. Perkel JM. The sleight-of-hand trick that can simplify scientific computing. Nature. 2023;617: 212–213.

67. Courtes L. Reproducibility and performance: Why choose? Comput Sci Eng. 2022;24: 77–80.

68. GNU Project Volunteers. GNU guix reference manual. <https://guix.gnu.org/manual/en/html_node/Invoking-guix-pack.html>; 2022.

69. Wurmus R, Uyar B, Osberg B, Franke V, Gosdschan A, Wreczycka K, et al. PiGx: Reproducible genomics analysis pipelines with GNU guix. Gigascience. 2018;7: giy123.

70. Perkel JM. Challenge to scientists: Does your ten-year-old code still run? Nature. 2020;584: 656–658.

71. Tuloup J. JupyterLite: Jupyter WebAssembly python. 2021. Available: <https://blog.jupyter.org/jupyterlite-jupyter-%EF%B8%8F-webassembly-%EF%B8%8F-python-f6e2e41ab3fa>

72. Balamuta JJ. Quarto-webr: Quarto extension to embed webr for html documents. 2023. Available: <https://github.com/coatless/quarto-webr>

73. Veiga Leprevost F da, Grüning BA, Alves Aflitos S, Röst HL, Uszkoreit J, Barsnes H, et al. BioContainers: An open-source and community-driven framework for software standardization. Bioinformatics. 2017;33: 2580–2582.

74. Archmiller AA, Johnson AD, Nolan J, Edwards M, Elliott LH, Ferguson JM, et al. Computational reproducibility in the wildlife society’s flagship journals. J Wildl Manage. 2020;84: 1012–1017.

75. McKiernan EC, Bourne PE, Brown CT, Buck S, Kenall A, Lin J, et al. How open science helps researchers succeed. Elife. 2016;5.

76. UNESCO recommendation on open science. 2023. Available: <https://www.unesco.org/en/open-science/about>

77. Wilkinson MD, Dumontier M, Aalbersberg IJJ, Appleton G, Axton M, Baak A, et al. The FAIR guiding principles for scientific data management and stewardship. Sci Data. 2016;3: 160018.

78. Miyakawa T. No raw data, no science: Another possible source of the reproducibility crisis. Mol Brain. 2020;13: 24.

79. Donoho DL. An invitation to reproducible computational research. Biostatistics. 2010;11: 385–388.

80. Gabelica M, Bojčić R, Puljak L. Many researchers were not compliant with their published data sharing statement: A mixed-methods study. J Clin Epidemiol. 2022;150: 33–41.

81. Brazma A, Hingamp P, Quackenbush J, Sherlock G, Spellman P, Stoeckert C, et al. Minimum information about a microarray experiment (MIAME)-toward standards for microarray data. Nat Genet. 2001;29: 365–371.

82. National Health and Medical Research Council. Management of Data and Information in Research: A guide supporting the Australian Code for the Responsible Conduct of Research. 2019. Available: <https://www.nhmrc.gov.au/sites/default/files/documents/attachments/Management-of-Data-and-Information-in-Research.pdf>

83. Office of The Director, National Institutes of Health. Final NIH policy for data management and sharing. 2023.

84. Packer M. Data sharing in medical research. BMJ. 2018;360: k510.

85. Hanson B, Sugden A, Alberts B. Making data maximally available. Science. 2011;331: 649.

86. Cheifet B. Open data in a deeply connected world. Genome Biol. 2020;21: 96.

87. Venkatesaramani R, Malin BA, Vorobeychik Y. Re-identification of individuals in genomic datasets using public face images. Sci Adv. 2021;7: eabg3296.

88. Schadt EE, Woo S, Hao K. Bayesian method to predict individual SNP genotypes from gene expression data. Nat Genet. 2012;44: 603–608.

89. Gymrek M, McGuire AL, Golan D, Halperin E, Erlich Y. Identifying personal genomes by surname inference. Science. 2013;339: 321–324.

90. Kalendralis P, Sloep M, Soest J van, Dekker A, Fijten R. Making radiotherapy more efficient with FAIR data. Phys Med. 2021;82: 158–162.

91. Edgar R, Domrachev M, Lash AE. Gene expression omnibus: NCBI gene expression and hybridization array data repository. Nucleic Acids Res. 2002;30: 207–210.

92. Kodama Y, Shumway M, Leinonen R, International Nucleotide Sequence Database Collaboration. The sequence read archive: Explosive growth of sequencing data. Nucleic Acids Res. 2012;40: D54–6.

93. Deutsch EW, Csordas A, Sun Z, Jarnuczak A, Perez-Riverol Y, Ternent T, et al. The ProteomeXchange consortium in 2017: Supporting the cultural change in proteomics public data deposition. Nucleic Acids Res. 2017;45: D1100–D1106.

94. Pampel H, Vierkant P, Scholze F, Bertelmann R, Kindling M, Klump J, et al. Making research data repositories visible: The re3data.org registry. PLoS One. 2013;8: e78080.

95. Roche DG, Kruuk LEB, Lanfear R, Binning SA. Public data archiving in ecology and evolution: How well are we doing? PLoS Biol. 2015;13: e1002295.

96. Hennessey J, Ge S. A cross disciplinary study of link decay and the effectiveness of mitigation techniques. BMC Bioinformatics. 2013;14 Suppl 14: S5.

97. Khatri P, Sirota M, Butte AJ. Ten years of pathway analysis: Current approaches and outstanding challenges. PLoS Comput Biol. 2012;8: e1002375.

98. Nekrutenko A, Taylor J. Next-generation sequencing data interpretation: Enhancing reproducibility and accessibility. Nature Reviews Genetics. 2012;13: 667–672. doi:[10.1038/nrg3305](https://doi.org/10.1038/nrg3305)

99. Simoneau J, Dumontier S, Gosselin R, Scott MS. Current RNA-seq methodology reporting limits reproducibility. Brief Bioinform. 2021;22: 140–145.

100. Mellor DT, Chambers K, Collings A, Graf C, Kiermer V, Macleod MR, et al. MDAR: Towards minimum reporting standards for life scientists. Open Science Framework; 2022.

101. Rule A, Birmingham A, Zuniga C, Altintas I, Huang S-C, Knight R, et al. Ten simple rules for writing and sharing computational analyses in jupyter notebooks. PLoS Comput Biol. 2019;15: e1007007.

102. Ziemann M, Bora A. A recipe for extremely reproducible enrichment analysis v1. protocols.io. 2023.

103. Barba LA. The hard road to reproducibility. Science. 2016;354: 142.

104. The Turing Way Community, Arnold B, Bowler L, Gibson S, Herterich P, Higman R, et al. The turing way: A handbook for reproducible data science. Zenodo; 2019. doi:<https://doi.org/10.5281/zenodo.3233986>

105. Beaulieu-Jones BK, Greene CS. Reproducibility of computational workflows is automated using continuous analysis. Nat Biotechnol. 2017;35: 342–346.

106. Blischak JD, Carbonetto P, Stephens M. Creating and sharing reproducible research code the workflowr way. F1000Res. 2019;8: 1749.

107. Huang Y, Gottardo R. Comparability and reproducibility of biomedical data. Brief Bioinform. 2013;14: 391–401.

108. Kaelin WG Jr. Publish houses of brick, not mansions of straw. Nature. 2017;545: 387.

109. Munafò MR, Nosek BA, Bishop DVM, Button KS, Chambers CD, Sert NP du, et al. A manifesto for reproducible science. Nat Hum Behav. 2017;1: 0021.

110. Diong J, Kroeger CM, Reynolds KJ, Barnett A, Bero LA. Strengthening the incentives for responsible research practices in Australian health and medical research funding. Res Integr Peer Rev. 2021;6: 11.

111. Williams JJ, Tractenberg RE, Batut B, Becker EA, Brown AM, Burke ML, et al. Optimizing short-format training: An international consensus on effective, inclusive, and career-spanning professional development in the life sciences and beyond. bioRxiv. 2023. p. 2023.03.10.531570.

112. Wilson G. Software carpentry: Lessons learned. F1000Res. 2014;3: 62.